COLUMN

シリコンバレーに学ぶキャリア・デベロップメント

第3回

成功の秘けつ



東原朋成

筆者の友人で，ベンチャ企業の創業者兼Vice Presidentでもある K 氏の机の上には，**図1**のような“成功の秘密（秘けつ）”が置いてあります．彼はこれを Fortune誌の広告から入手し，この信条（処世訓）を毎朝読み返しているそうです．

こういった信条を飾っている人は大勢います．筆者の元上司もオフィスに“Success”なる信条を示したポスタを飾っていました．Fortune誌やForbes誌のようなビジネス誌では，こういった信条をポスタや机上の置き物にして販売している会社の広告をよく見かけます．

図1の10項目を見てみると，なんだかあたりまえのことが列記されていると思われるかもしれません．10番目がいちばん重要だということになるのですが，これにしてもずいぶんあたりまえのことと感じるでしょう．意訳すると「頼りにされる人間であれ．それにはつねに正直（率直）であれ」ということです．実際，上司や同僚，部下，顧客，投資家に頼りにされないのでは成功はおぼつかないように思います．

The New York Times誌でベストセラーのアワードを受けた“The One Minute Millionaire”（Mark Victor Hansen, Robert G. Allen著，Harmony Books；日本語版は『ドリーム【小説篇】』/『ドリーム【実践篇】』，徳間書店）という書籍があります．この書の言わんとするところをひと言で言い表すなら，「いい仲間を手に入れろ」となるのではないかと思います．先ほどの10項目の10番目と非常に関連があります．お互い頼りになるから良き仲間になれるのです．

筆者は，本誌で幾度となく人脈（ネットワーク）について紹介してきました．シリコンバレーは文字どおり“ネットワーク”社会と言うことができます．簡単に図式化すると**図2**のようになります．良き仲間を得てその仲間と会社を創業する，仲間と会社を移っていく，以前の仲間（上司，部下も仲間）に引き抜かれる…．筆者の元上司はこれまで三つの会社を設立していますが（うち1社は上場までこぎつけた），創業のコア・メンバはいつも同じです．彼にとってこのコア・メンバは必須だと言えるのでしょう．良い仲間に出会ったら関係をたいせつに保つことが重要です．

Vice Presidentが部下のDirectorを外部から雇おうとしたとき，だれを雇いたいかという質問に対して，自分のために働いてくれた信頼のおける以前の部下がいちばんの候補とよく言われます．また，自分の上司と非常にうまが合い，働きやすかったとなれば，その上司がほかの会社に移った場合に自分もいっしょに移りたいと思うのもごく自然なことです．キャリア・アップも人

10 secrets to success

1. How you think is everything;
2. Decide upon your true dreams and goals;
3. Take action;
4. Never stop learning;
5. Be persistent and work hard(smart);
6. Learn to analyze details;
7. Focus your time and money;
8. Don't be afraid to innovate: Be different;
9. Deal and communicate with people effectively;
10. Be honest and dependable: Take responsibility
—Otherwise all other items won't matter.

10項目を整理すると，
- 目標，目的をつねに持て．
- その目標，目的に向かって一生懸命努力しろ．
- その目標，その目的に向けてつねに学び続けろ．現在の自分，状況を見極めることを学べ．
- その目標，目的から目をそらすな．
- ほかと違う自分を恐れるな．
- 頼りにされる人間たれ．

図1　成功の秘けつ

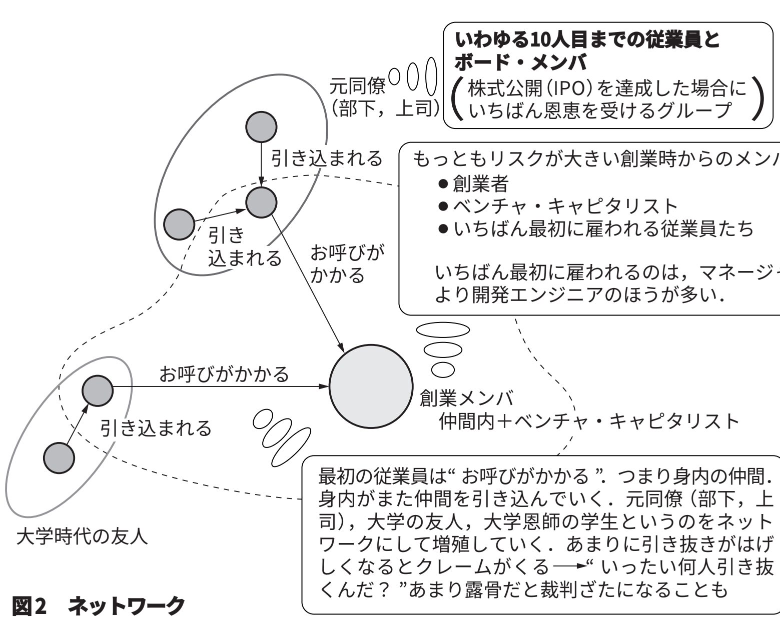


図2　ネットワーク

脈（ネットワーク）が非常に重要になります．前回述べたジョブ・ホッパではネットワークを築く機会が限られてしまいます．

Stanford UniversityなどでMBAの学位をとるエンジニアやマネージャもたくさんいます．卒業生の一致した意見は，学位や学業そのものよりも，その膨大な卒業生ネットワーク（学閥とも言えるのかもしれないが，日本社会の学閥とは趣きが異なる）がもっとも役立っているとのことです．

ところでネットワークとしては，二つのタイプを持つことが必要だとされています．密で小規模な（仲間）ネットワークと，粗で大規模なネットワークです．最近ではLinkedIn（http://www.linkedin.com/）など，ネットワークを作るためのツールも豊富にあります．こういったツールを使ってネットワークの輪を広げておくことも，キャリア・アップを考える際には必要なことと言えます．

インタビュー模擬体験

答 前回の解答（2005年4月号，pp.84-85）

問題

A社とB社は競合関係にあります．両社とも新聞，雑誌への広告合戦によって利益を上げることが可能ですが，その効果は表1のとおりとします．

さて，こういった予想の場合，A社とB社には決定的な戦略があるのでしょうか？ またその場合の予想利益は？

表1　広告の方法と効果

広告の方法	利益	
	A社	B社
A社，B社ともに新聞広告を出す	8億円	9億円
A社は新聞に，B社は雑誌に広告を出す	7億円	8億円
A社は雑誌に，B社は新聞に広告を出す	9億円	8億円
A社，B社とも雑誌に広告を出す	8億円	7億円

問題はゲーム理論の応用でした．この問題は単純化してありますが，ゲーム理論をかいま見ることができます．

問題に示した例では，Aは雑誌に，Bは新聞に広告を出すのが得策です．Aが新聞に広告を出した場合，Bの出方によって利益は8億円か7億円ですが，雑誌に広告を出した場合は9億円もしくは8億円といずれの場合も利益が増えます．Bについても同様に考え，新聞に広告を出すほうが良い結果を生むことがわかります．整理すると，

　A：雑誌に広告を出す→予想される利益は9億円
　B：新聞に広告を出す→予想される利益は8億円

であり，A，Bには決定的な（お互い他社に左右されない）広告戦略があると言えます．

この例は決定的戦略がある場合でした．例えばこの問題が，A，Bとも雑誌に広告を出した場合にBの利益が9億円になったとしたらどうでしょうか．この場合，BはAの動きを予測しなくてはならなくなります．Aの動きを予測でき，Aは雑誌に広告を出すはず（この場合）とわかれば，Bは雑誌に広告を出すべきです．そのほうが利益が増えるからです．したがって，Bの立場は決定的とは言えません．Aの動きを予測する，つまりまずAの立場になる必要が出てきます．実際のビジネスでは，相手の動きによって戦略が変わってくることがよくあります．

ゲーム理論（ノーベル賞学者のJohn Nashがパイオニア）を学ぶと，このあたりのことを理解できます．映画“A Beautiful Mind”（邦題：ビューティフル・マインド）でもゲーム理論が出てきます．企業のマーケティング部門などでは，複雑な予測モデルをもとに商品計画を立案しているとも聞きます．MBAコースでは，ゲーム理論はかならず勉強する必須課目の一つです．

● 前回のエクササイズについて

Balance Sheetなどを見てみましたか？　会社が提供するこれらの資料には出てこない項目がいくつもあります．投資の専門家は隠れた（隠された）数字を読み取ることが要求されます．

隠れた数字と言えば，例えば研究開発費はコストとして計上されますが，会社の資産には組み込まれません．研究開発を適切に資産として計上することが必要なのかもしれません．議論の分かれるところのようです．筆者の元上司は，部門のBalance SheetやIncome Statementを良く見せる技術にたけていました．これだけでエンジニアリング・マネージャからビジネス・マネージャに昇格したくらいです．

問 今月の問題

問題

二つのプロジェクトの提案A，Bがあります．A，BのNPV（Net Present Value）を求め，どちらの提案がNPVの観点から良い提案と言えるかを評価してください．

ここでDiscount Rate（簡単に言うと，お金の未来の価値を現在の価値で評価するときのレート．例えば10%とすると，1年後の1万円は現在の価値では9千円ということになる）を10%とします．

A：初期投資は5億円．2年目以降の5年間，毎年2.5億円の利益がある．

B：初期投資は6億円．3年目以降10年にわたり，毎年1.8億円の利益がある．

提案されたプロジェクトを評価する手段の一つにNPVがあります．この場合のNPVとは，簡単に言うと，プロジェクトへの

リスト1 このコードをシミュレーションすると…

```
module race;

reg a, c, d, e, f, j;

wire b;

initial begin
    a = 1'b0;
    c = 1'b0;
    d = 1'b0;
    e = 1'b0;
    f = 1'b0;
    j = 1'b0;

    #1;
    printvalue();

    c = 1'b1;
    e = 1'b1;
    f = 1'b1;
    j = 1'b1;

    #1;
    printvalue();
end

always @(b or c or e or f)begin
    a = b & c;          // Line A
    d = e & f;          // Line B
end

assign b = d & j;       // Line C

task printvalue;
    begin
    $display("===== OUTPUTS =====");
    $display("a = %h", a");
    $display("b = %h", b");
    $display("c = %h", c");
    $display("d = %h", d");
    $display("e = %h", e");
    $display("f = %h", f");
    $display("j = %h", j");
    end
endtask

endmodule
```

リスト2 レース・コンディションをなくすための変更(1)

```
always @(b or c or e or f)begin
    a <= b & c;         // Line A
    d <= e & f;         // Line B
end
```

リスト3 レース・コンディションをなくす変更(2)

```
always @(b or c)begin
   a = b & c;
end

always @(e or f)begin
   d = e & f;
end
```

投資と将来得られる利益を現在価値に置き換えて評価したプロジェクトの価値です．NPVがマイナスになるということは，投資を将来回収できないということであり，特別な理由がない限り良い投資（プロジェクト）とは言えません．複数の提案がある場合，NPVを比較することもあります．

これはCorporate Financeというコースでかならず勉強する内容です．また，プロジェクト・マネージャには，この程度の経済知識は要求されます．

エクササイズ

エンジニアのままキャリア・アップを図るために，シニア（リーダ・レベル）検証エンジニアを目ざす方のための練習問題です．

リスト1に示すVerilog HDLソース・コードをシミュレーションすると，結果はどうなるのでしょうか？

レース・コンディション（信号の競合状態）になるという方がいるかもしれません．レースにならなかったという場合も，もちろんあります．**リスト1**のコードには，レース条件が潜在します．ただし，レースになるかどうかはシミュレータによります．シミュレータに依存するレースが生じると，容易には原因を特定できません．

シミュレータがA，B，Cの順に実行すればレースになりません．しかし，Verilog HDLは，always文を途中で抜けてほかの行を実行し，またalways文に戻ることを許しています．つまり，A，C，Bの順に実行することも許されます．そのほうが実行速度を上げられるとシミュレータが判断すれば，この順番で実行することはなんら不思議ではありません．実際，A，C，Bと実行された場合，このコードはレース条件になります．ここで，bはalways外で変化しますが，always文を途中で抜けて戻ったというケースではbの変化によるalwaysの再実行は行われません．これは，alwaysのセンシティビティ・リストのチェックはそのalways文が実行中でないときに行われますが，途中で抜けてまた戻ったというような場合は"実行中でない"という条件に合致しないためです．

それでは派生問題です．レース条件を生み出さないように**リスト1**のコードを変更してください．

ここでは二つの例を示します．**リスト2**はalways文中でノンブロッキング代入を使いました．しかし，コーディング・ガイドラインでは，組み合わせ論理をalways文中で実現するにはブロッキング代入を使うとされているのがふつうです．こういったガイドラインはもちろん有効です．しかし，いかなるコードにも適用できるものではありません．本質を理解しないまま うのみにするのは危険です．

もっとも，組み合わせ論理をalways文中で実現するのにノンブロッキング代入を使いたくない方も多いはずです．その場合，**リスト3**のように変更すればレースを回避することが可能です．この場合，always文の数が増えます．このことはシミュレーション時間に影響を与える可能性があります．

優秀な設計エンジニアなら**リスト1**のようなコードを避けるものです．ただし，大規模な設計では，後から追加，削除した論理によってこの例のようなケースを作ってしまうこともあります．このような例の場合，bを継続的代入で実現する必要があるのか？という観点に立ち戻ってコードを検討してみる必要があります．

コーディング・ガイドラインを守って設計することはもちろんですが，そのガイドラインが前提としているものを理解したうえでそれを守る必要があります．設計グループの検証を担当するリーダは，シミュレータをよく理解していることが求められます．検証エンジニアが先頭に立ってガイドラインを作成し，設計エンジニアをトレーニングする必要があります．

さらなる派生問題もあります．ノンブロッキング代入を使えばレース条件を回避できると説明しました．その根拠について考えてみてください．

とうはら・ともなり